RESTful web services are the first step to developing great microservices.

Web Development

It is well suited Spring module for web application development. We can easily create a self-contained HTTP server using embedded Tomcat, Jetty or Undertow. We can use the spring-boot- starter-web module to start and running application quickly.

SpringApplication

It is a class which provides the convenient way to bootstrap a spring application which can be started from main method. You can call start your application just by calling a static run() method.

1. **public** **static** **void** main(String[] args){
2. SpringApplication.run(className.**class**, args);
3. }

## Application Events and Listeners

Spring Boot uses events to handle variety of tasks. It allows us to create factories file that are used to add listeners. we can refer it by using ApplicationListener key.

Always create factories file in META-INF folder like: **META-INF/spring.factories**

## Admin Support

Spring Boot provides the facility to enable admin related features for the application. It is used to access and manage application remotely. We can enable it by simply using spring.application.admin.enabled property.

## Externalized Configuration

Spring Boot allows us to externalize our configuration so that we can work with the same application in different environments. Application use YAML files to externalize configuration.

## Properties Files

Spring Boot provides rich set of Application Properties. So, we can use that in properties file of our project. Properties file is used to set properties like: **server-port = 8082** and many others. It helps to organize application properties.

## YAML Support

It provides convenient way for specifying hierarchical configuration. It is a superset of JSON. The SpringApplication class automatically support YAML. It is successful alternative of properties.

## Type-safe Configuration

Strong type-safe configuration is provided to govern and validate the configuration of application. Application configuration is always a crucial task which should be type-safe. We can also use annotation provided by this library.

## Logging

Spring Boot uses Common logging for all internal logging. Logging dependencies are managed by default. We should not change logging dependencies, if there is no required customization is needed.

## Security

Spring Boot applications are spring bases web applications. So, it is secure by default with basic authentication on all HTTP endpoints. A rich set of Endpoints are available for develop a secure Spring Boot application.

We need to configure it in order to make it a Spring Boot project. Here, we are adding parent to our Maven project. It is used to declare that our project is a child to this parent project.

1. **<parent>**
2. **<groupId>**org.springframework.boot**</groupId>**
3. **<artifactId>**spring-boot-starter-parent**</artifactId>**
4. **<version>**1.4.2.RELEASE**</version>**
5. **</parent>**

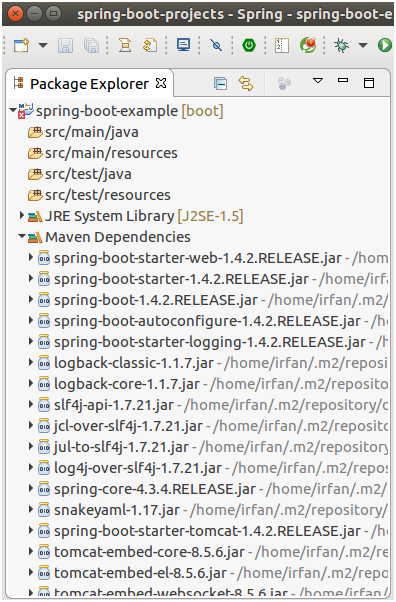
#### Note - Maven project will add web dependency to the project by downloading the jar.

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.springframework.boot**</groupId>**
4. **<artifactId>**spring-boot-starter-web**</artifactId>**
5. **</dependency>**
6. **</dependencies>**

After that add Java version for the project.

1. **<properties>**
2. **<java.version>**1.8**</java.version>**
3. **</properties>**

Now, our project should have the following directly structure. You can notice that maven has created a new dependency folder to store the jar files.



**SpringBootExample.java**

1. **package** com.javatpoint;
3. **import** org.springframework.boot.SpringApplication;
4. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
6. @SpringBootApplication
7. **public** **class** SpringBootExample {
8. **public** **static** **void** main(String[] args) {
9. SpringApplication.run(SpringBootExample.**class**, args);
10. }
11. }

Now, run this class as a Java Application.

The SpringApplication, inside the main is a Spring Boot class. It is used to bootstrap our application. It calls static method run() which takes two arguments, one is class type and second is string array. It starts auto-configured tomcat web server when Spring application is started.

#### Note - The @RestController and @RequestMapping annotations are Spring MVC annotations. They are not specific to Spring Boot.

1. **import** org.springframework.web.bind.annotation.RequestMapping;
2. **import** org.springframework.web.bind.annotation.RestController;
3. @RestController
4. **public** **class** HomeController {
5. @RequestMapping(value = "/hello", method = "GET")
6. **public** String hello(){
7. **return**"Hello!";
8. }
9. }

A GET request by **localhost:8080/hello** url will map to the hello() method.

# Dependency Management

Spring Boot manages dependencies and configuration automatically. You don't need to specify version for any of that dependencies.

Spring Boot upgrades all dependencies automatically when you upgrade Spring Boot.

#### Note - We can also specify a version by overriding Spring Boot recommendations if we think that's necessary.

#### Note - Each release of Spring Boot is associated with a base version of the Spring Framework so we highly recommend you to not specify its version on your own.

## Inheriting the starter parent

We can configure our project to inherit from the **spring-boot-starter-parent** by simply setting as below.

1. **<parent>**
2. **<groupId>**org.springframework.boot**</groupId>**
3. **<artifactId>**spring-boot-starter-parent**</artifactId>**
4. **<version>**2.0.0.BUILD-SNAPSHOT**</version>**
5. **</parent>**

#### Note - You should only need to specify the Spring Boot version number on this dependency. If you import additional starters, you can safely exclude the version number.

Adding Spring Boot Maven Plugin

We can include Maven plugin in our pom.xml file. It is used to package the project as an executable jar. We are adding it here.

1. **<build>**
2. **<plugins>**
3. **<plugin>**
4. **<groupId>**org.springframework.boot**</groupId>**
5. **<artifactId>**spring-boot-maven-plugin**</artifactId>**
6. **</plugin>**
7. **</plugins>**
8. **</build>**

# Spring Boot Application Properties

Spring Boot provides various properties which can be specified inside our project's **application.properties** file. These properties have default values and you can set that inside the properties file. Properties are used to set values like: server-port number, database connection configuration etc.

The following table contains common Spring Boot properties.

|  |  |  |
| --- | --- | --- |
| **Property** | **Default value** | **Description** |
| banner.charset | UTF-8 | It is used to set banner file encoding. |
| banner.location | classpath:banner.txt | It is used to set banner file location. |
| logging.file |  | It is used to set log file name. For example data.log. |
| spring.application.index |  | It is used to set application index. |
| spring.application.name |  | It is used to set application name. |
| spring.application.admin.enabled | false | It is used to enable admin features for the application. |
| spring.config.location |  | It is used to config file locations. |
| spring.config.name | application | It is used to set config file name. |
| spring.mail.default-encoding | UTF-8 | It is used to set default MimeMessage encoding. |
| spring.mail.host |  | It is used to set SMTP server host. For example smtp.example.com. |
| spring.mail.password |  | It is used to set login password of the SMTP server. |
| spring.mail.port |  | It is used to set SMTP server port. |
| spring.mail.test-connection | false | It is used to test that the mail server is available on startup. |
| spring.mail.username |  | It is used to set login user of the SMTP server. |
| spring.main.sources |  | It is used to set sources for the application. |
| server.address |  | It is used to set network address to which the server should bind to. |
| server.connection-timeout |  | It is used to set time in milliseconds that connectors will wait for another HTTP request before closing the connection. |
| server.context-path |  | It is used to set context path of the application. |
| server.port | 8080 | It is used to set HTTP port. |
| server.server-header |  | It is used for the Server response header (no header is sent if empty) |
| server.servlet-path | / | It is used to set path of the main dispatcher servlet |
| server.ssl.enabled |  | It is used to enable SSL support. |
| spring.http.multipart.enabled | True | It is used to enable support of multi-part uploads. |
| spring.http.multipart.max-file-size | 1MB | It is used to set max file size. |
| spring.mvc.async.request-timeout |  | It is used to set time in milliseconds. |
| spring.mvc.date-format |  | It is used to set date format. For example dd/MM/yyyy. |
| spring.mvc.locale |  | It is used to set locale for the application. |
| spring.social.facebook.app-id |  | It is used to set application's Facebook App ID. |
| spring.social.linkedin.app-id |  | It is used to set application's LinkedIn App ID. |
| spring.social.twitter.app-id |  | It is used to set application's Twitter App ID. |
| security.basic.authorize-mode | role | It is used to set security authorize mode to apply. |
| security.basic.enabled | true | It is used to enable basic authentication. |

# Spring Boot Starters

Starters are a set of convenient dependency descriptors which we can include in our application.

Spring Boot provides built-in starters which makes development easier and rapid. For example, if we want to get started using Spring and JPA for database access, just include the **spring-boot-starter-data-jpa** dependency in your project.

Starter should follow a naming pattern like: **spring-boot-starter**-\*, where \* is a particular type of application. This naming structure is intended to help when you need to find a starter.

The following application starters are provided by Spring Boot under the org.springframework.boot group:

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-thymeleaf | It is used to build MVC web applications using Thymeleaf views. |
| spring-boot-starter-data-couchbase | This is used for Couchbase document-oriented database and Spring Data Couchbase. |
| spring-boot-starter-artemis | It is used for JMS messaging using Apache Artemis. |
| spring-boot-starter-web-services | It is used for Spring Web Services. |
| spring-boot-starter-mail | It is used to support Java Mail and Spring Framework's email sending. |
| spring-boot-starter-data-redis | It is used for Redis key-value data store with Spring Data Redis and the Jedis client. |
| spring-boot-starter-web | It is used for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container. |
| spring-boot-starter-data-gemfire | It is used to GemFire distributed data store and Spring Data GemFire. |
| spring-boot-starter-activemq | It is used to JMS messaging using Apache ActiveMQ. |
| spring-boot-starter-data-elasticsearch | It is used to Elasticsearch search and analytics engine and Spring Data Elasticsearch. |
| spring-boot-starter-integration | It is used for Spring Integration. |
| spring-boot-starter-test | It is used to test Spring Boot applications with libraries including JUnit, Hamcrest and Mockito. |
| spring-boot-starter-jdbc | It is used for JDBC with the Tomcat JDBC connection pool. |
| spring-boot-starter-mobile | It is used for building web applications using Spring Mobile. |
| spring-boot-starter-validation | It is used for Java Bean Validation with Hibernate Validator. |
| spring-boot-starter-hateoas | It is used to build hypermedia-based RESTful web application with Spring MVC and Spring HATEOAS. |
| spring-boot-starter-jersey | It is used to build RESTful web applications using JAX-RS and Jersey. An alternative to spring-boot-starter-web. |
| spring-boot-starter-data-neo4j | It is used for Neo4j graph database and Spring Data Neo4j. |
| spring-boot-starter-data-ldap | It is used for Spring Data LDAP. |
| spring-boot-starter-websocket | It is used for building WebSocket. applications using Spring Framework?s WebSocket support. |
| spring-boot-starter-aop | It is used for aspect-oriented programming with Spring AOP and AspectJ. |
| spring-boot-starter-amqp | It is used for Spring AMQP and Rabbit MQ. |
| spring-boot-starter-data-cassandra | It is used for Cassandra distributed database and Spring Data Cassandra. |
| spring-boot-starter-social-facebook | It is used for Spring Social Facebook. |
| spring-boot-starter-jta-atomikos | It is used for JTA transactions using Atomikos. |
| spring-boot-starter-security | It is used for Spring Security. |
| spring-boot-starter-mustache | It is used for building MVC web applications using Mustache views. |
| spring-boot-starter-data-jpa | It is used for Spring Data JPA with Hibernate. |
| spring-boot-starter | It is used for core starter, including auto-configuration support, logging and YAML. |
| spring-boot-starter-groovy-templates | It is used for building MVC web applications using Groovy Templates views. |
| spring-boot-starter-freemarker | It is used for building MVC web applications using FreeMarker views. |
| spring-boot-starter-batch | It is used for Spring Batch. |
| spring-boot-starter-social-linkedin | It is used for Spring Social LinkedIn. |
| spring-boot-starter-cache | It is used for Spring Framework?s caching support. |
| spring-boot-starter-data-solr | It is used for the Apache Solr search platform with Spring Data Solr. |
| spring-boot-starter-data-mongodb | It is used for MongoDB document-oriented database and Spring Data MongoDB. |
| spring-boot-starter-jooq | It is used for jOOQ to access SQL databases. An alternative to spring-boot-starter-data-jpa or spring-boot-starter-jdbc. |
| spring-boot-starter-jta-narayana | It is used for Spring Boot Narayana JTA Starter. |
| spring-boot-starter-cloud-connectors | It is used for Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku. |
| spring-boot-starter-jta-bitronix | It is used for JTA transactions using Bitronix. |
| spring-boot-starter-social-twitter | It is used for Spring Social Twitter. |
| spring-boot-starter-data-rest | It is used for exposing Spring Data repositories over REST using Spring Data REST. |

## Spring Boot production starters

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-actuator | It is used for Spring Boot?s Actuator which provides production ready features to help you monitor and manage your application. |
| spring-boot-starter-remote-shell | It is used for the CRaSH remote shell to monitor and manage your application over SSH. Deprecated since 1.5. |

## Spring Boot technical starters

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-undertow | It is used for Undertow as the embedded servlet container. An alternative to spring-boot-starter-tomcat. |
| spring-boot-starter-jetty | It is used for Jetty as the embedded servlet container. An alternative to spring-boot-starter-tomcat. |
| spring-boot-starter-logging | It is used for logging using Logback. Default logging starter. |
| spring-boot-starter-tomcat | It is used for Tomcat as the embedded servlet container. Default servlet container starter used by spring-boot-starter-web. |
| spring-boot-starter-log4j2 | It is used for Log4j2 for logging. An alternative to spring-boot-starter-logging. |

# Spring Boot Actuator

Spring Boot provides actuator to monitor and manage our application. Actuator is a tool which has HTTP endpoints. when application is pushed to production, you can choose to manage and monitor your application using HTTP endpoints.

To get production-ready features, we should use spring-boot-actuator module. We can enable this feature by adding it to the pom.xml file.

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.springframework.boot**</groupId>**
4. **<artifactId>**spring-boot-starter-actuator**</artifactId>**
5. **</dependency>**
6. **</dependencies>**

## Endpoints

Actuator endpoints allow us to monitor and interact with our Spring Boot application. Spring Boot includes number of built-in endpoints and we can also add custom.

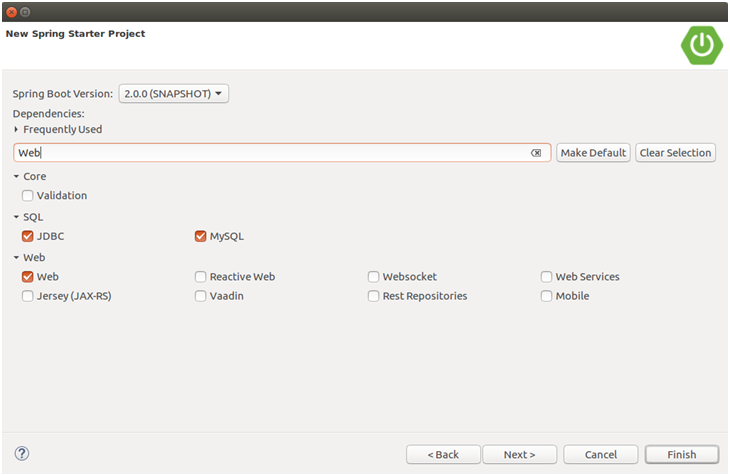
The following table contains the available endpoints.

|  |  |  |
| --- | --- | --- |
| **Id** | **Description** | **Sensitive default** |
| actuator | It provides a hypermedia-based "discovery page" for the other endpoints. It requires Spring HATEOAS to be on the classpath. | True |
| auditevents | It exposes audit events information for the current application. | True |
| autoconfig | It is used to display an auto-configuration report showing all auto-configuration candidates and the reason why they 'were' or 'were not' applied. | True |
| beans | It is used to display a complete list of all the Spring beans in your application. | True |
| configprops | It is used to display a collated list of all @ConfigurationProperties. | True |
| dump | It is used to perform a thread dump. | True |
| env | It is used to expose properties from Spring's ConfigurableEnvironment. | True |
| flyway | It is used to show any Flyway database migrations that have been applied. | True |
| health | It is used to show application health information. | False |
| info | It is used to display arbitrary application info. | False |
| loggers | It is used to show and modify the configuration of loggers in the application. | True |
| liquibase | It is used to show any Liquibase database migrations that have been applied. | True |
| Metrics | It is used to show metrics information for the current application. | True |
| mappings | It is used to display a collated list of all @RequestMapping paths. | True |
| shutdown | It is used to allow the application to be gracefully shutdown. | True |
| trace | It is used to display trace information. | True |

For Spring MVC, the following additional endpoints can also be used.

|  |  |  |
| --- | --- | --- |
| **Id** | **Description** | **Sensitive default** |
| docs | It is used to display documentation, including example requests and responses for the Actuator's endpoints. | False |
| heapdump | It is used to return a GZip compressed hprof heap dump file. | True |
| jolokia | It is used to expose JMX beans over HTTP (when Jolokia is on the classpath). | True |
| logfile | It is used to return the contents of the logfile. |  |

# Spring Boot JDBC



Configure database into application.properties file.

// application.properties

1. spring.datasource.url=jdbc:mysql://localhost:3306/springbootdb
2. spring.datasource.username=root
3. spring.datasource.password=mysql
4. spring.jpa.hibernate.ddl-auto=create-drop
5. **import** org.springframework.web.bind.annotation.RequestMapping;
6. **import** org.springframework.beans.factory.annotation.Autowired;
7. **import** org.springframework.jdbc.core.JdbcTemplate;
8. **import** org.springframework.web.bind.annotation.RestController;
9. @RestController
10. **public** **class** SpringBootJdbcController {
11. @Autowired
12. JdbcTemplate jdbc;
13. @RequestMapping("/insert")
14. **public** String index(){
15. jdbc.execute("insert into user(name,email)values('javatpoint','java@javatpoint.com')");
16. **return**"data inserted Successfully";
17. }
18. }

**why it's considered opinionated?** Well because it makes a judgment on its own. Sometimes it imports things which you don't want, but don't worry, Spring Boot also provides ways to override auto-configuration settings.

The [@EnableAutoConfiguration](http://www.java67.com/2018/05/difference-between-springbootapplication-vs-EnableAutoConfiguration-annotations-Spring-Boot.html) is used to enable auto-configuration but @SpringBootApplication does a lot more than that.  
  
It also combines @Configuration and @ComponentScan annotations to enable Java-based configuration and component scanning in your project.  
  
The [@SpringBootApplication](https://javarevisited.blogspot.sg/2018/05/the-springbootapplication-annotation-example-java-spring-boot.html) is in fact combination of @Configuration, @ComponentScan and @EnableAutoConfiguration annotations

**Spring Actuator** is another cool Spring Boot feature which allows seeing inside a running application.  
  
Yes, you read it correctly. It allows you to see inside an application. Since Spring Boot is all about auto-configuration it makes debugging difficult and at some point in time, you want to know which [beans](https://javarevisited.blogspot.com/2012/05/what-is-bean-scope-in-spring-mvc.html#axzz5IZi1jCsQ) are created in Spring's Application Context and how Controllers are mapped. Spring Actuator provides all that information.  
  
It provides several endpoints e.g. a REST endpoint to retrieve this kind of information over the web. It also provides a lot of insight and metrics about application health e.g. [CPU and memory usage](http://javarevisited.blogspot.sg/2013/06/find-cpu-and-memory-used-by-java-solaris-prstat-command-example.html), number of threads etc.  
  
It also comes with a remote shell which you can use to securely go inside Spring Boot application and run some command to expose the same set of data. You can even use JMX to control this behavior at runtime.  
  
Btw, it's important to secure your Spring Actuator endpoints because it exposes a lot of confidential information and a potentially dangerous one-two. For example, by using /showdown endpoint you can kill a Spring Boot application.  
  
But, don't worry. You can use Spring Security to secure Spring Actuator endpoints.   
  
  
  
It's also disabled by default and you need to use either @SpringBootApplication or @EnableAutoConfiguration annotations on the Main class to enable the auto-configuration feature.

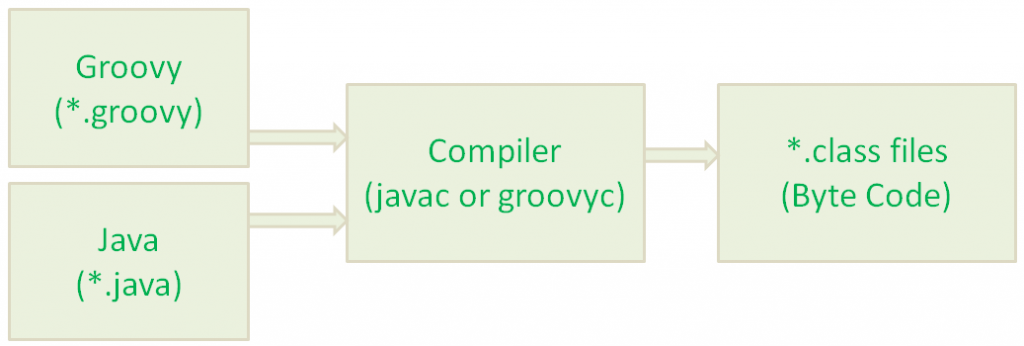
**Where do you define properties in Spring Boot application?**  
You can define both application and Spring boot related properties into a file called application.properties. You can create this file manually or you can use Spring Initializer to create this file, albeit empty.  
  
You don't need to do any special configuration to instruct Spring Boot load this file. If it exists in classpath then Spring Boot automatically loads it and configure itself and application code according.  
  
**Can you change the port of Embedded Tomcat server in Spring boot? If Yes, How?**  
Yes, we can change the port of Embedded Tomcat Server in Spring Boot by adding a property called server.port in the application.properties file.  
  
**What is the difference between an embedded container and a WAR?**  
The main difference between an embedded container and a WAR file is that you can Spring Boot application as a JAR from the command prompt without setting up a web server. But to run a WAR file, you need to first set up a [web server](http://www.java67.com/2016/06/3-difference-between-web-server-vs-application-server-vs-servlet-container.html) like Tomcat which has Servlet container and then you need to deploy WAR there.  
  
**What embedded containers does Spring Boot support?**  
Spring Boot support three embedded containers: Tomcat, Jetty, and Undertow. By default, it uses Tomcat as embedded containers but you can change it to Jetty or Undertow.  
  
**What are some common Spring Boot annotations?**  
Some of the most common Spring Boot annotations are @EnableAutoConfiguration, @SpringBootApplication, @SpringBootConfiguration, and @SpringBootTest.  
  
The @EnableAutoConfiguration is used to enable auto-configuration on Spring Boot application, while @SpringBootApplication is used on the [Main class](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html) to allow it to run a JAR file. @SpringBootTest is used to run unit test on Spring Boot environment.  
  
  
**Can you name some common Spring Boot Starter POMs?**  
Some of the most common Spring Boot Start dependencies or POMs are spring-boot-starter, spring-boot-starter-web, spring-boot-starter-test. You can use spring-boot-starter-web to enable Spring MVC in Spring Boot application.  
  
  
  
**Can you control logging with Spring Boot? How?**  
Yes, we can control logging with Spring Boot by specifying log levels on application.properties file. Spring Boot loads this file when it exists in the [classpath](http://www.java67.com/2012/08/what-is-path-and-classpath-in-java-difference.html) and it can be used to configure both Spring Boot and application code.  
  
Spring Boot uses Commons Logging for all internal logging and you can change log levels by adding following lines in the application.properties file:  
  
logging.level.org.springframework=DEBUG  
logging.level.com.demo=INFO

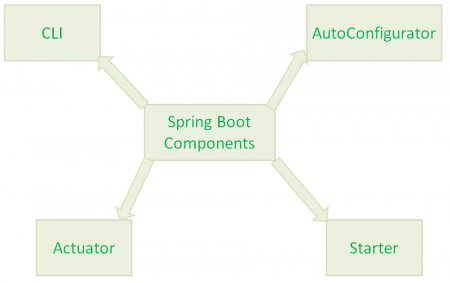
### Why Spring Boot?

* To ease the Java-based applications Development, Unit Test and Integration Test Process.
* To reduce Development, Unit Test and Integration Test time by providing some defaults.
* To increase Productivity.

<https://www.journaldev.com/7969/spring-boot-tutorial>

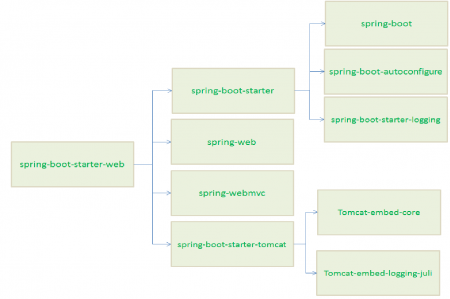
Spring Boot Framework Programming model is inspired by Groovy Programming model. Spring Boot internally uses some Groovy based techniques and tools to provide default imports and configuration.





Spring Boot Starter component combines all related jars into single jar file so that we can add only jar file dependency to our build files. Instead of adding above 4 jars files to our build file, we need to add one and only one jar file: “spring-boot-starter-web” jar file.

When we add “spring-boot-starter-web” jar file dependency to our build file, then Spring Boot Framework will automatically download all required jars and add to our project classpath.

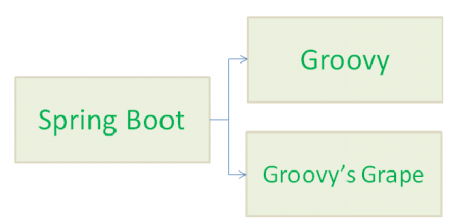


The main responsibility of Spring Boot AutoConfigurator is to reduce the Spring Configuration. If we develop Spring applications in Spring Boot,then We dont need to define single XML configuration and almost no or minimal Annotation configuration. Spring Boot AutoConfigurator component will take care of providing those information.

SpringBootApplicationAnnotation

* Spring Boot Starter reduces build’s dependencies and Spring Boot AutoConfigurator reduces the Spring Configuration.
* As we discussed that Spring Boot Starter has a dependency on Spring Boot AutoConfigurator, Spring Boot Starter triggers Spring Boot AutoConfigurator automatically.

Spring Boot Framework internally uses Groovy to add some defaults like Default import statements, Application main() method etc. When we run Groovy Scripts from CLI Command prompt, it uses this main() method to run the Spring Boot Application.

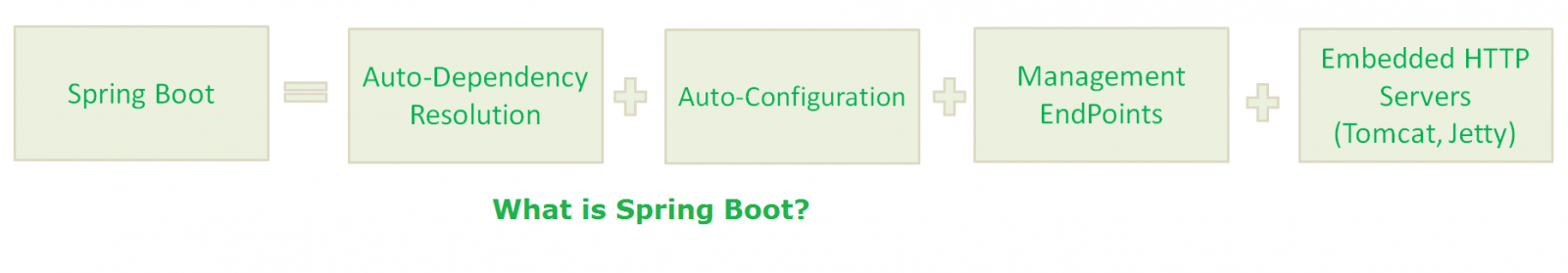


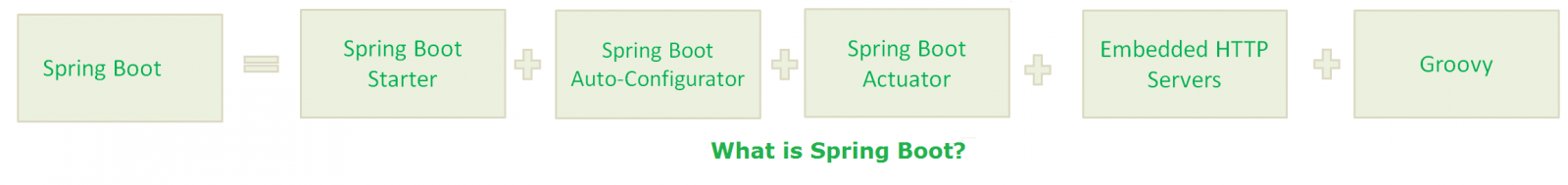
who will provide all these things to our Spring Boot HelloWorld application?

* No imports
* No other XML configuration to define Spring MVC Components like Views,ViewResolver etc.
* No web.xml and No DispatcherServlet declaration
* No build scripts to create our Application war file
* No need to build war file to deploy this application

Answer to this question: It is the responsibility of Spring Boot Core Components, Groovy Compiler (groovyc) and Groovy Grape (Groovy’s JAR Dependency Manager).

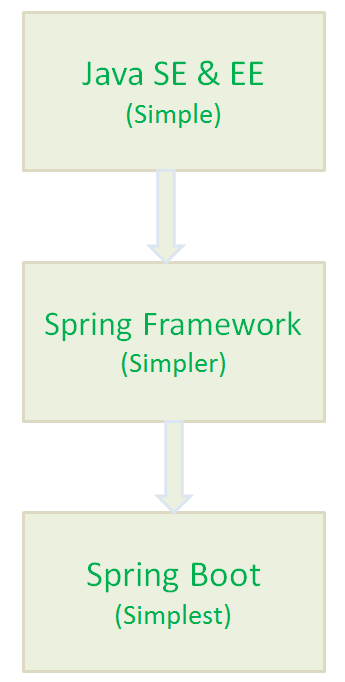
Spring Boot Components uses Groovy Compiler and Groovy Grape to provide some Defaults lime adding required imports, providing required configuration, resolving jar dependencies, adding main() method etc. As a Spring Boot Developer, We don’t need to worry all these things. Spring Boot Framework will take care of all these things for us.





### Why we need Spring Boot?

1. [Spring Framework](https://www.journaldev.com/16922/spring-framework) aims to simplify Java Applications Development.
2. Spring Boot Framework aims to simplify Spring Development.



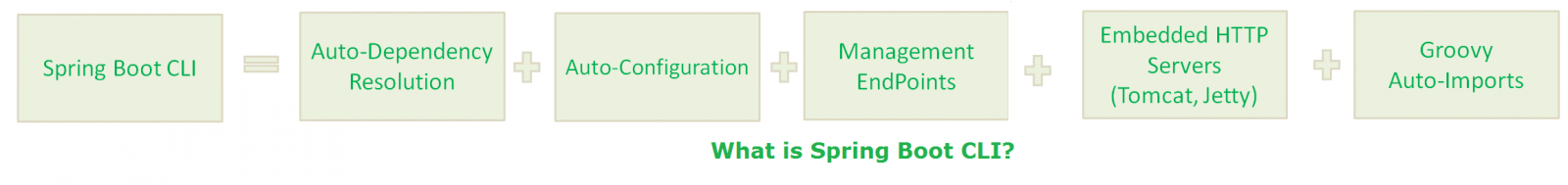
Spring Boot Starters are just JAR Files. They are used by Spring Boot Framework to provide “Auto-Dependency Resolution”.

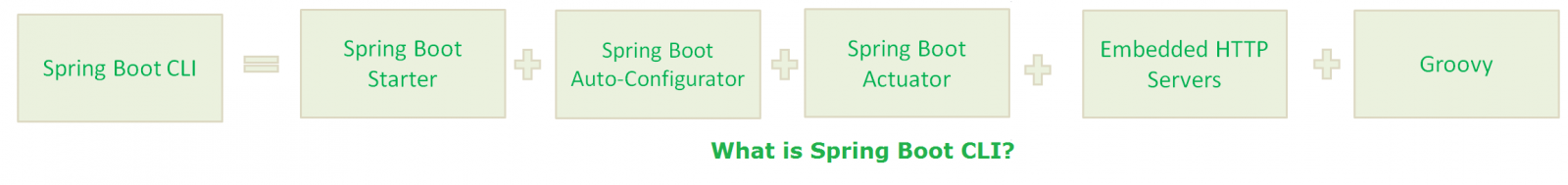
Spring Boot AutoConfigurator is used by Spring Boot Framework to provide “Auto-Configuration”.

Spring Boot Actuator is used by Spring Boot Framework to provide “Management EndPoints” to see Application Internals, Metrics etc.

### What is Spring Boot CLI?

In simple words, Spring Boot CLI is Auto Dependency Resolution, Auto-Configuration, Management EndPoints, Embedded HTTP Servers(Jetty, Tomcat etc.) and (Groovy, Auto-Imports)





With Spring Boot CLI:

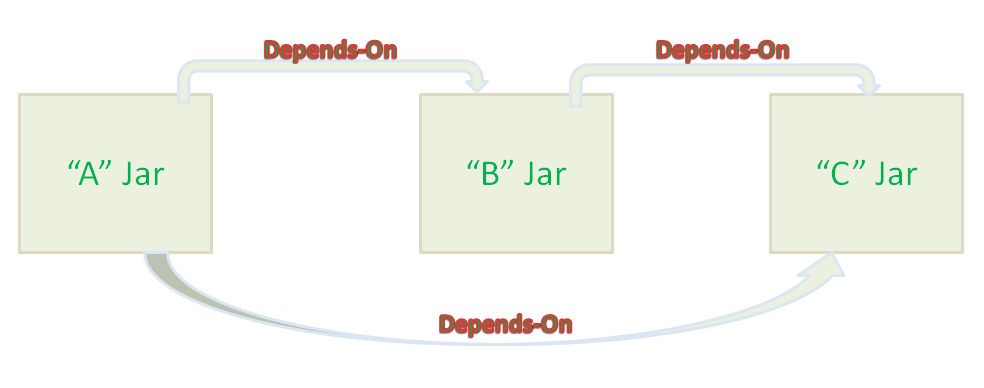
* No Semicolons
* No Public and private access modifiers
* No Imports(Most)
* No “return” statement
* No setters and getters
* No Application class with main() method(It takes care by SpringApplication class).
* No Gradle/Maven builds.
* No separate HTTP Servers.

### Spring Boot With Maven/Gradle?

Spring Boot Framework uses one of the greatest features of Maven/Gradle build tools: **“Transitively Dependency Resolution Management”**.

### What is “Transitively Dependency Resolution Management”?

“Transitively Dependency Resolution Management” means: If we define an “A” dependency in build scripts, “A” is dependent on “B” and “B” is dependent on “C”, That means “A” is also dependent on “C”.  
Then Build Tools will download and add all Three Jar files “A”, “B” and “C” to our application classpath.



Read more: <http://www.java67.com/2018/06/top-15-spring-boot-interview-questions-answers-java-jee-programmers.html#ixzz5i8UKxqYb>

<https://www.javatpoint.com/spring-interview-questions>

<http://www.springboottutorial.com/spring-boot-vs-spring-mvc-vs-spring> (Read)

*Most important feature of Spring Framework is Dependency Injection. At the core of all Spring Modules is Dependency Injection or IOC Inversion of Control.*

when DI or IOC is used properly, we can develop loosely coupled applications. And loosely coupled applications can be easily unit tested.

*Spring MVC Framework provides decoupled way of developing web applications. With simple concepts like Dispatcher Servlet, ModelAndView and View Resolver, it makes it easy to develop web applications.*

<https://www.journaldev.com/18156/spring-boot-mongodb>

@RequestMapping(value = "/{userId}", method = RequestMethod.GET)

public User getUser(@PathVariable String userId) {

LOG.info("Getting user with ID: {}.", userId);

return userRepository.findOne(userId);

}

@RequestMapping(value = "/create", method = RequestMethod.POST)

public User addNewUsers(@RequestBody User user) {

LOG.info("Saving user.");

return userRepository.save(user);

}

## MongoTemplate vs MongoRepository

* MongoTemplate provides a lot more control when it comes to querying data and what data to pull from database.
* Spring Data repositories provide us a convenient outlook on how to fetch data.
* MongoTemplate is database dependent. What this means is, with Spring Data repositories, you can easily switch to a different database altogether by simply using a different Spring Data repositories for MySQL or Neo4J or anything else. This is not possible with MongoTemplate.

# Redis Config

spring.cache.type=redis

spring.redis.host=localhost

spring.redis.port=6379

Also, use the @EnableCaching annotation on Spring Boot main class:

Now, to put something into the cache, we use @Cacheable annotation:

@Cacheable(value = "users", key = "#userId", unless = "#result.followers < 12000")

@RequestMapping(value = "/{userId}", method = RequestMethod.GET)

public User getUser(@PathVariable String userId) {

LOG.info("Getting user with ID {}.", userId);

return userRepository.findOne(Long.valueOf(userId));

}

### Updating Cache

Cache values should also update whenever their actual objects value are updated. This can be done using @CachePut annotation:

@CachePut(value = "users", key = "#user.id")

@PutMapping("/update")

public User updatePersonByID(@RequestBody User user) {

userRepository.save(user);

return user;

}

### Clearing Cache

If some data is to be deleted from actual Database, there won’t be a point to keep it in cache anymore. We can clear cache data using @CacheEvict annotation:

@CacheEvict(value = "users", allEntries=true)

@DeleteMapping("/{id}")

public void deleteUserByID(@PathVariable Long id) {

LOG.info("deleting person with id {}", id);

userRepository.delete(id);

}

## Redis Cache Limits

Although Redis is very fast, it still has no limits on storing any amount of data on a 64-bit system. It can only store 3GB of data on a 32-bit system. More available memory can result into a more hit ratio but this will tend to cease once too much memory is occupied by Redis.  
When cache size reaches the memory limit, old data is removed to make place for new one.